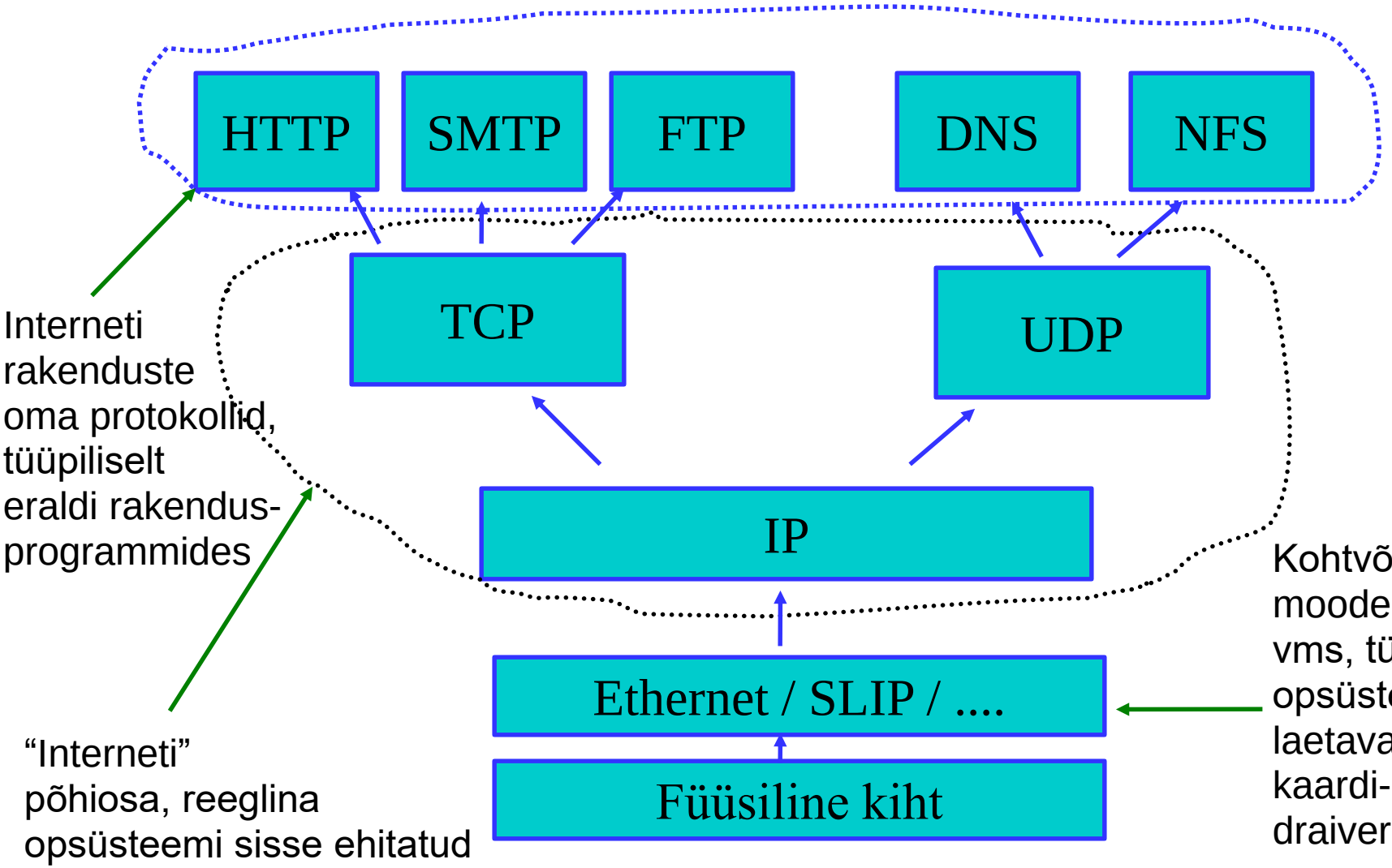
Sissejuhatus infotehnoloogiasse

interneti rakenduste tehnoloogia

Loengu ülevaade. Interneti rakendused.

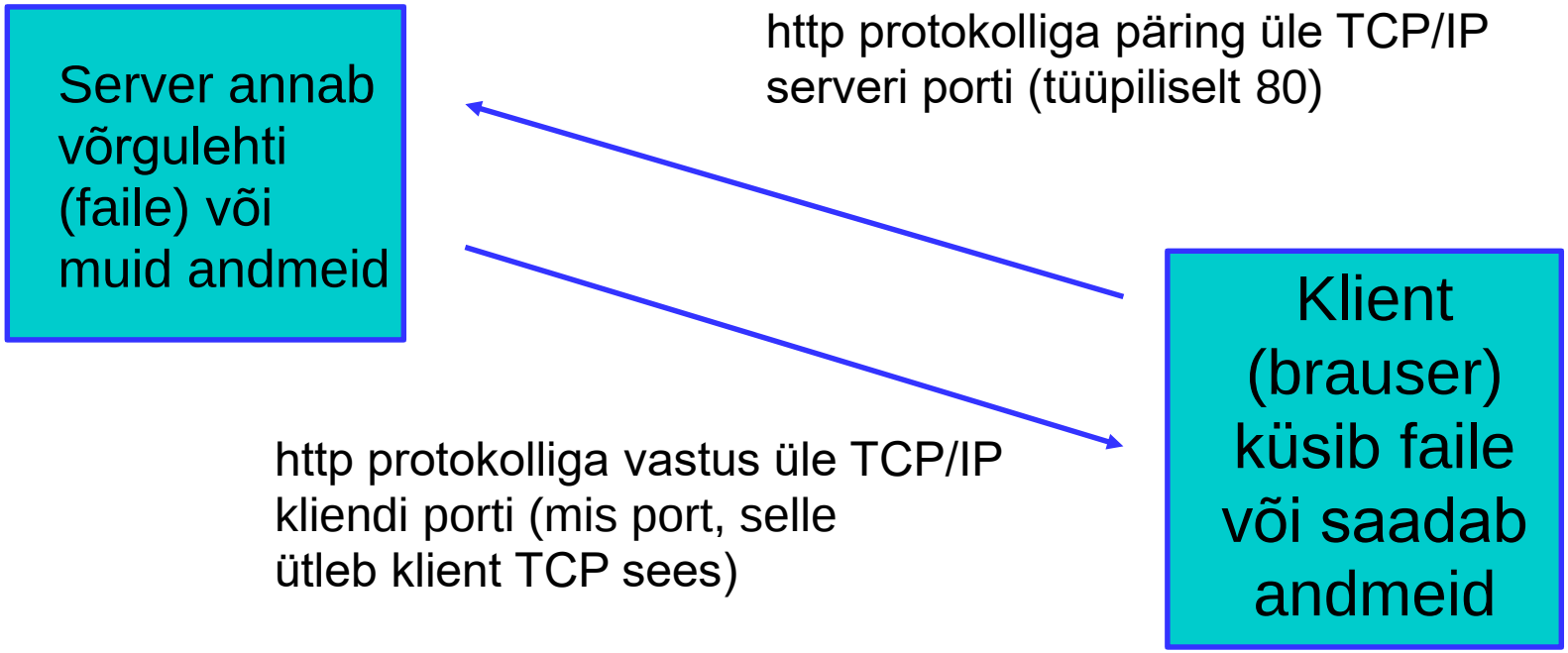
- Meeldetuletus: baasprotokollid
- HTTP protokoll
- Failide tõmbamine ja serveerimine:
  - Kuidas http abil faili tõmmata
  - Info saatmine serverile: cgi protokoll
- Mis brauseril sees on:
  - HTML
  - CSS
  - Javascript
- Klassikaline veebirakendus: server ehitab html teksti
- Single-page app:
  - server ehitab datat: json teksti
  - javascript brauseris muudab/ehitab html teksti otse brauseri kõhus
- Masintöödeldav andmete esitamine: JSON ja XML
- Tehnoloogiastack

Kokkuvõte protokollindusest internetis



HTTP ühendused: failide küsimine ja nende andmine

- **HTTP on omaette protokoll TCP peal**, mida kasutatakse veebilehtede, piltide, tekstifailide, zip failide, jsoni jne jne saatmiseks veebiserveri ja brauseri vahel.



Milline on HTTP protokollis päring?

HTTP päring on esmajoones tekstiline käsk serverile:
“anna mulle selline fail”, kus näidatakse ära:

- konkreetne küsimus-käsk
- faili asukoht ja nimi
- protokoll, mida küsija kasutab
- ja soovi korral lisainfot, nagu küsija programmi tüüp (chrome, firefox, curl, ...)

Samas saab HTTP päringule kaasa anda teksti kujul andmeid,
mida server siis salvestab ja/või rehkendab nende järgi vastuse

Milline on HTTP protokollis vastus?

HTTP ei ole ehitatud “biti või baidi” tasemel, vaid teksti ridade kaupa: päis, tühi rida, tekstiread.

päis

```
HTTP/1.1 200 OK
Date: Thu, 06 Nov 2003 13:50:07 GMT
Server: Apache/1.3.19 (Unix) PHP/4.1.1
Last-Modified: Sat, 10 Apr 1999 09:29:18 GMT
ETag: "46d8-297-370f19ee"
Accept-Ranges: bytes
Content-Length: 663
Connection: close
Content-Type: text/html
```

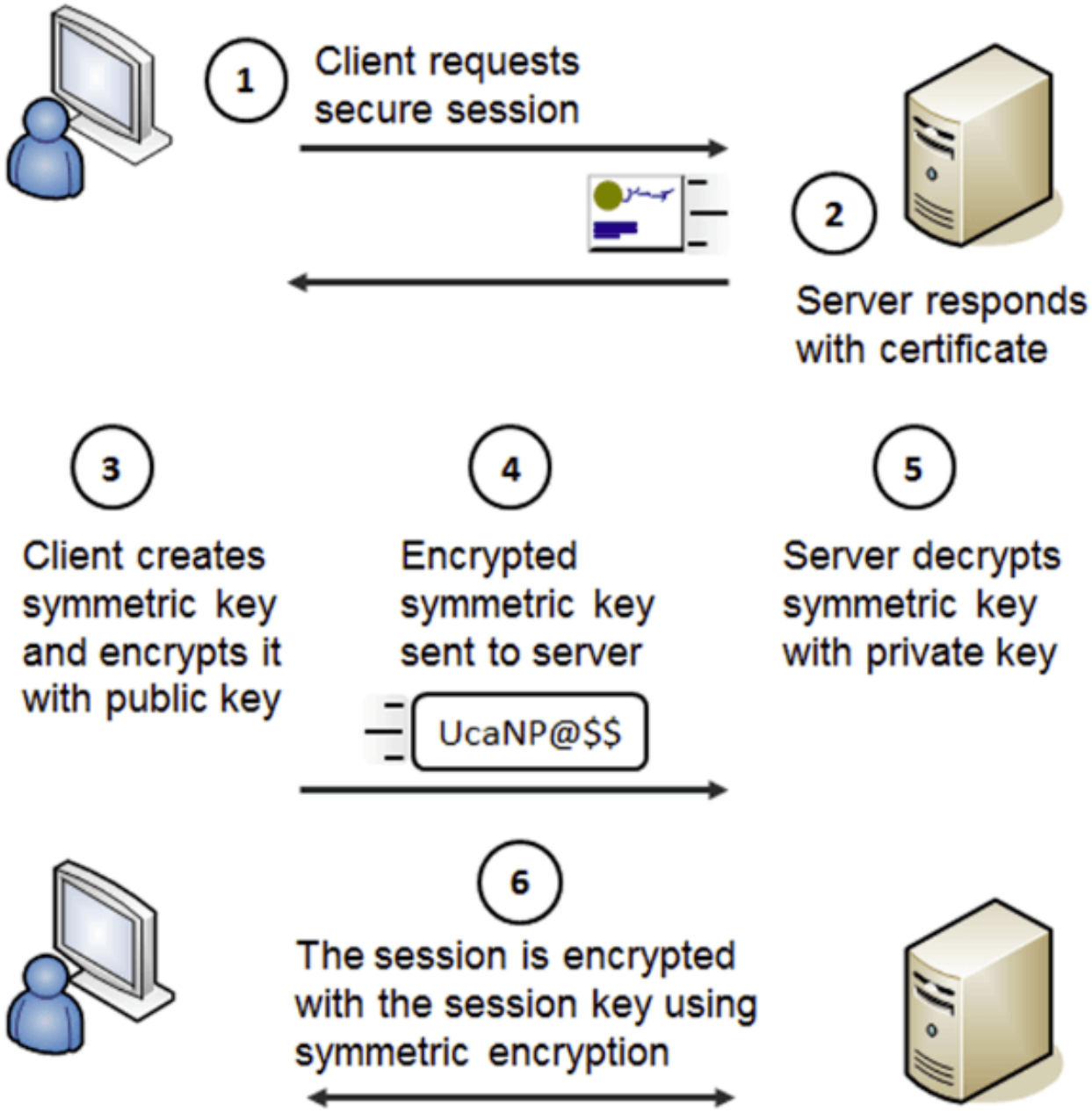
tühi rida

Tegelik sisu

```
<html>
 <head>
  <META HTTP-EQUIV="Content-type" CONTENT="text/html;
   charset=ISO-8859-1">
 .....
```

Mis on https?

HTTPS päring on krüpteeritud variant http päringust:
päring ise ja vastus krüpteeritakse võtmega,
mis automaatselt genereeritakse iga ühenduse jaoks.


1
Client requests
secure session
2
Server responds
with certificate
3
Client creates
symmetric key
and encrypts it
with public key
4
Encrypted
symmetric key
sent to server
UcaNP@$$
5
Server decrypts
symmetric key
with private key
6
The session is encrypted
with the session key using
symmetric encryption

Kuidas html-faili tõmmata?

- käsurealt curl programmiga (teine variant salvestab http päise):

curl https://news.ycombinator.com

curl https://news.ycombinator.com -i > ajut.txt

- pythoni programmist:

```
import urllib.request
req = urllib.request.urlopen('http://news.ycombinator.com')
html = req.read()
print(html)
```

Html: veebilehtede põhikeel

- Kujunduskeel, mitte programmeerimiskeel
- Hea tutorial: https://www.w3schools.com/html/
- Veebirakenduste kursus meil:

  http://lambda.ee/wiki/V%C3%B5rgurakendused_I

Tutvume põhimõtetega loengus tehtavate näidete abil

Veebilehtede (failide) serveerimine

- Veebilehed on lihtsalt tekstid, mille ette server paneb http-päise
- Veebileht võetakse reeglina kas:
  - olemasoleva failina arvuti kettal
  - või tekstina, mille teeb iga kord uuesti mõni programm:
    - Programm võib olla külge-ehitatud tava-veebiserverile (php)
    - Või käia kohe eri-veebiserveri sees (java tomcat)
    - Või olla eraldi programm, mille server käivitab
    - Kombinatsioonid serverite ahelatega on samuti levinud

Käivitame lihtsa serveri: flask (flask.pocoo.org/)

- Installi:

sudo apt-get install python3

sudo apt-get install python3-flask

- Tee programm hello.py:

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello():
        return "Hello World!“
```

- Käivita:

FLASK_APP=hello.py flask run

- Vaata brauserist:

http://127.0.0.1:5000/

Serveeri faile flaski serveri abil

- Teeme väikesed katsefailid katse.txt ja katse.html

- Flask serveerib faile kataloogist **static**, teeme selle ja kopeerime:

```
mkdir static
cd static
cp /mnt/c/Users/Tanel/katse.txt .
cp /mnt/c/Users/Tanel/katse.html .
```

- Vaata brauserist:

http://127.0.0.1:5000/static/katse.txt
http://127.0.0.1:5000/static/katse.html

Andmete saatmine serverile: cgi konventsioon

- URL-i lõpus (peale ?) oleva teksti saab serveri kutsutav programm kätte
- Brauser saadab vormi info cgi-kodeeritult nii: a=1&b=35 jne, kus a ja b jne on välja nimed, võrduse järel aga kasutaja sisestatud väärtused
- Tüüpiline url cgi parameetritega

http://dijkstra.cs.ttu.ee/~tammet/cgi-bin/loeng/tst6.py?a=1&b=2

http://dijkstra.cs.ttu.ee/~tammet/cgi-bin/loeng/tst8.py?a=12&b=44

- Vt lisaks ja detaile: http://lambda.ee/wiki/Cgi_examples

data.py

- Tee programm data.py:

```
from flask import Flask
from flask import request
app = Flask(__name__)
@app.route("/")
def hello():
        return "Hello World!"
@app.route('/cgi/show', methods=['POST', 'GET'])
def cgi():
  inparams={}
  keys=request.args.keys()
  for key in keys:
    inparams[key]=request.args.get(key)
  return(str(inparams))
```

- Vaata brauserist:

http://127.0.0.1:5000/cgi/show?a=1&b=2

sum.py

```
from flask import Flask
from flask import request
app = Flask(__name__)

@app.route('/cgi/sum', methods=['POST', 'GET'])
def sum():
  a=request.args.get(“a“,0)
  b=request.args.get(“b“,0)
  return(str(int(a)+int(b)))
```

http://127.0.0.1:5000/cgi/sum?a=1&b=2

Brauser oskab mitut keelt

- Iga normaalne brauser mõistab:

Põhiasjad:

- html
- css
- javascript

Lisatehnoloogiad:

- WebGL graafika
- Canvas
- Video ja audio
- Xslt
- Xml
- Väikesed andmebaasid
- ...

HTML

- Teksti paigutamise / lehe kujundamise keel
- Näited kohapeal
- Loe: http://www.w3schools.com/html
- Ametlik standard on HTML5:

  https://www.w3.org/TR/html5/

Klassikaline veebirakendus

- Server saadab brauserile html teksti
- Kasutaja täidab vormid/vajutab nuppe
- Brauser saadab serverile sisestatud parameetrid
- **Server arvutab parameetrite järgi uue html teksti**
- Brauser kuvab uue html teksti
- Jne ...

sumapp.py http://127.0.0.1:5000/cgi/sumapp

```
from flask import Flask
from flask import request
app = Flask(__name__)
@app.route('/cgi/sumapp', methods=['POST', 'GET'])
def myapp():
  a=request.args.get("a",0)
  b=request.args.get("b",0)
  r="sum is: "+str(int(a)+int(b))+"<p>"
  s="""
    get sum:<p>
    <form action='/cgi/sumapp'>
       a <input type='text' name='a'> <br>
       b <input type='text' name='b'> <p>
       <input type='submit' value='calculate!'>
     </form>
    """
  if a and b: res=r+s
  else: res=s
  return(res)
```

CSS

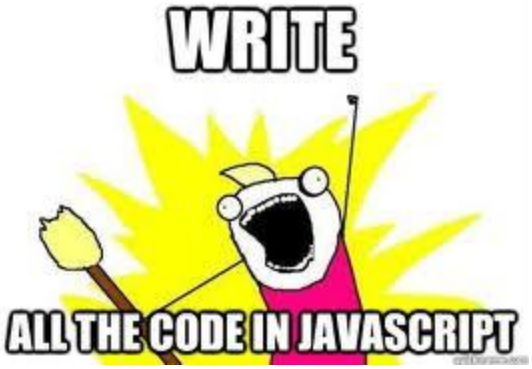
- **Täpset teksti paigutust ja kujundust võimaldav keel HTML täienduseks**

- Näited kohapeal
- Uuri lisaks: http://www.w3schools.com/css

Javascript

- **Brauseri programmeerimiskeel: javascripti programmid töötavad otse brauseris: muudavad htmli, css-i, võtavad ühendust serveriga jne jne**

- Näited kohapeal
- Uuri lisaks: http://www.w3schools.com/js



Single-page app

Wikipedia ütleb:

> A single-page application (SPA) is a web application or web site that interacts with the user by dynamically rewriting the current page rather than loading entire new pages from a server.
>
> This approach avoids interruption of the user experience between successive pages, making the application behave more like a desktop application.
>
> In an SPA, either all necessary code – HTML, JavaScript, and CSS – is retrieved with a single page load,[1] or the appropriate resources are dynamically loaded and added to the page as necessary

asum.html rehkendus javascriptis otse brauseris

```
<div id="ans"></div>
get sum:<p>
<form>
  a <input type='text' id='a'> <br>
  b <input type='text' id='b'> <p>
  <input type='button' onclick='calc()' value='calculate with javascript!'>
</form>
<script>
function calc() {
  var a,b,res;
  a=document.getElementById('a').value;
  b=document.getElementById('b').value;
  res=String(parseInt(a)+parseInt(b));
  document.getElementById('ans').innerHTML="Sum is: "+res;
}
</script>
```

AJAX: javascript asynchronous queries

AJAX tähistab: Javascript + asynchronous queries

Mõte selles, et saad brauseris javascriptiga avada serveri urli a la nii

```
fetch(url, {
   method: "get"
 }).then(r=>r.text()).then(handleresult);
```

ja siis brauseris saadud teksti või andmetega (tüüpiliselt javascripti formaadis ehk jsonis) edasi tegutseda.

Tähtis: seni, kui datat tõmmatakse, sinu javascripti programm ei hangu, vaid käib normaalselt edasi!

a2sum.html ajaxiga: rehkendus on serveris!

```
<div id="ans"></div>
get sum:<p>
<form>
  a <input type='text' id='a'> <br>
  b <input type='text' id='b'> <p>
  <input type='button' onclick='calc()' value='calculate with ajax!'>
</form>
<script>
function calc() {
  var a,b,url;
  a=document.getElementById('a').value;
  b=document.getElementById('b').value;
  url="/cgi/sum?a="+a+"&b="+b
  fetch(url, {
    method: "get"
  }).then(r=>r.text()).then(handleresult);
}
function handleresult(r) {
  document.getElementById('ans').innerHTML="Sum is: "+r;
}
</script>
```

JSON ja XML

- Eelmisel kümnendil muutus XML väga populaarseks andme-esituskeeleks.
- Seejärel hakati kasutama hulka keerulisi lisatehnoloogiad: XSLT, SOAP, XML Schema, UDDI jne, mis muutsid asja keeruliseks.
- Seejärel muutus populaarseks JSON kui hulga lihtsam andme-esituskeel: lihtsalt javascripti andmesüntaks, väga sarnane pythoni süntaksile a la

  {“nimi“: “Tanel“, “kursused“: [“itv0010“, “itv100“]}

- Praegu kasutatakse mõlemat, aga JSON on palju populaarsem. Laiatarbe-süsteemid a la Google pakuvad enamasti JSON-is andme-esitust

XML tähendab ...

- **eXtensible Markup Language**

<tootja><nimi>A Le Coq</nimi><linn>Tartu</linn></tootja>

- XML on:
  - Struktureeritud teksti esitamise formaat
  - XML standard ütleb, kuidas teksti struktuuri märgistada.
  - Saab kasutada andmete esitamiseks tekstina
  - Lihtne, aga veidi kohmakas kasutada

- XML ei ole:
  - Programmeerimiskeel.

JSON tähendab ...

- **Javascript Object Notation**

[1,2,“siin on tekst“, [“sisemine“,5], 10]
{“nimi“:“Peeter“, “pikkus“: 180, “hinded“: [5,3,2] }

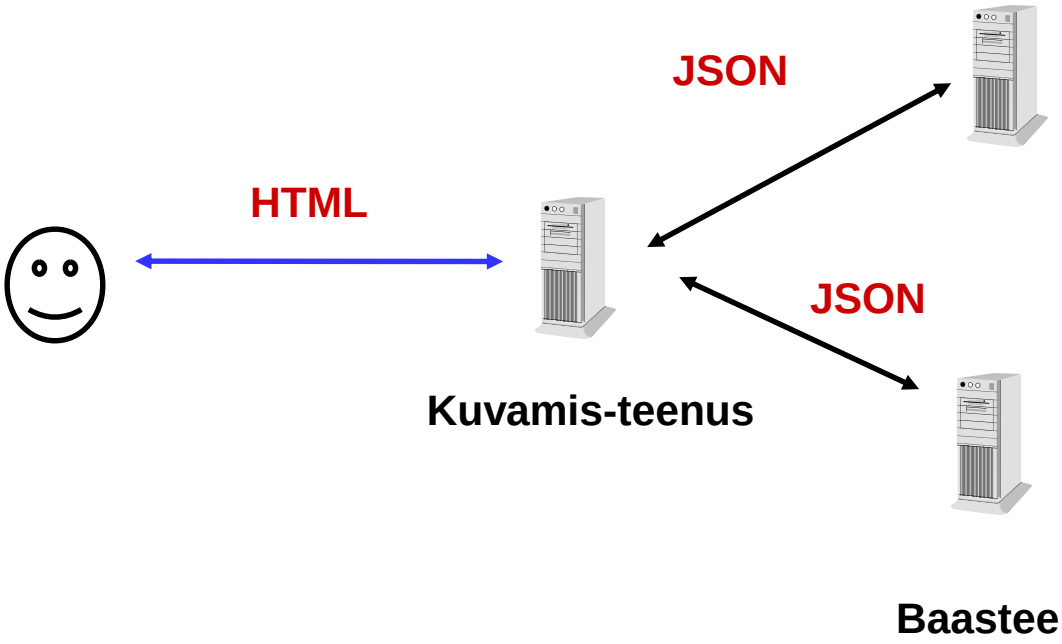
- JSON on:
  - Andmete esitamise formaat tekstina
  - Javascripti programmeerimiskeele „native“ andmestruktuur
  - Väga lihtne ja mugav kasutada
  - Kaasajal brauserirakendustes eelistatum kui XML.

- JSON ei ole:
  - Programmeerimiskeel.

Võrguteenused

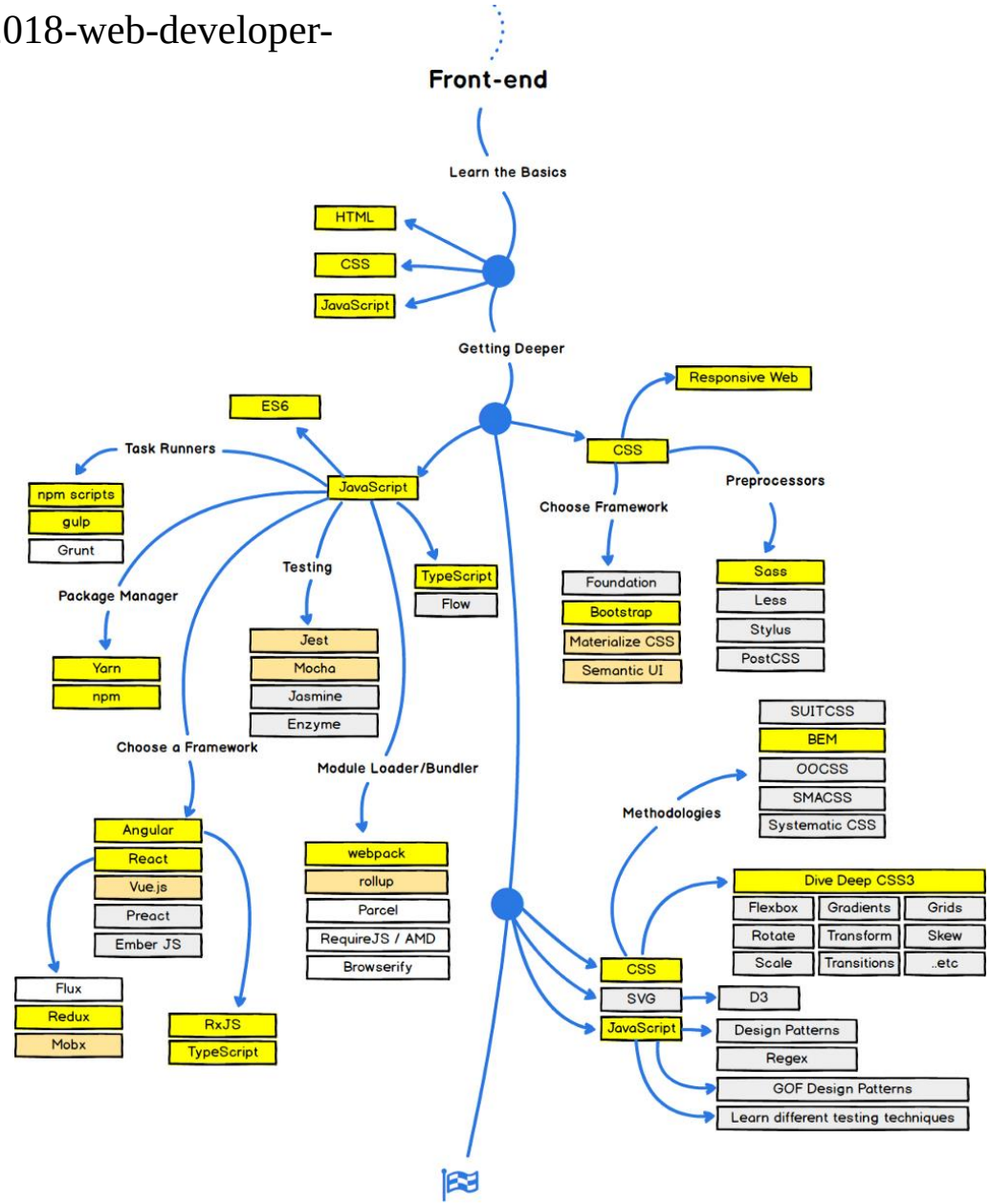
- Probleem: raske on teha programme, mis loeks teisest serverist HTML-lehekülgi ja leiaks nende sisust kergesti vajaliku info

- Lahendus: teeme võrgulehekülgi lihtsalt andmetekstina jsonis või xmlis nii et programm teises arvutis suudaks neid lugeda



Front-end tehnoloogiastack on päris keeruline:

https://codeburst.io/the-2018-web-developer-roadmap-826b1b806e8d



ja back-end (serverirakendus) samuti:

https://codeburst.io/the-2018-web-developer-roadmap-826b1b806e8d